TRASHIGIMIA E KLASAVE

KAPITULLI 7

Prof. Ass. Dr. Isak Shabani

Trashegimia

- Përdorimi i një klase të vetme për të modeluar dy ose më tepër entitete që janë të ngjashme por kanë edhe ndryshime nuk është një projekt i mirë.
- Mekanizmi inheritance - projekton dy ose më tepër entitete që janë të ndryshëm por ndajnë shumë tipare të përbashkëta.
- Së pari përcaktojmë një klasë që përmban tiparet e përbashkëta të entiteteve.
- Pastaj përcaktojmë klasat si zgjerim i klasës së përbashkët, të tilla që ato trashëgojnë cdo gjë nga klasa bazë.

Trashigimia dhe roli i saj në POO

• Përmes mekanizmit të trashëgimisë (ang. inheritance), anëtarët e klasës ekzistuese mund të shfrytëzohen gjatë definimit të klasave të reja.

• Klasa ekzistuese njihet si klasë bazë (ang. base class), kurse klasa e re i trashëgon karakteristikat e klasës bazë dhe quhet klasë e nxjerrë (ang. derived class).

• Klasa e re krijohet duke e plotësuar klasën ekzistuese me anëtarë të tjerë, qofshin ato anëtarë të dhënash ose funksione.

• Gjatë kësaj, klasa bazë nuk pëson asnjë ndryshim.

Trashigimia e klasave

• Kur definojmë një klasë, themi `class – emri`, mund të japim emrin e superklasës, `super – class – emri`, për një klasë.

•Në disa kontekste një super klasë (super class) quhet klasë bazë (base class).

class – modifikuesi **class** *class – emri: super – class –emri*
{
deklarimet
}

Sintaksa. Definimi i një klase në C# si një nënklasë e superklasës

• Emri super – class është dhënë pas dy pikave

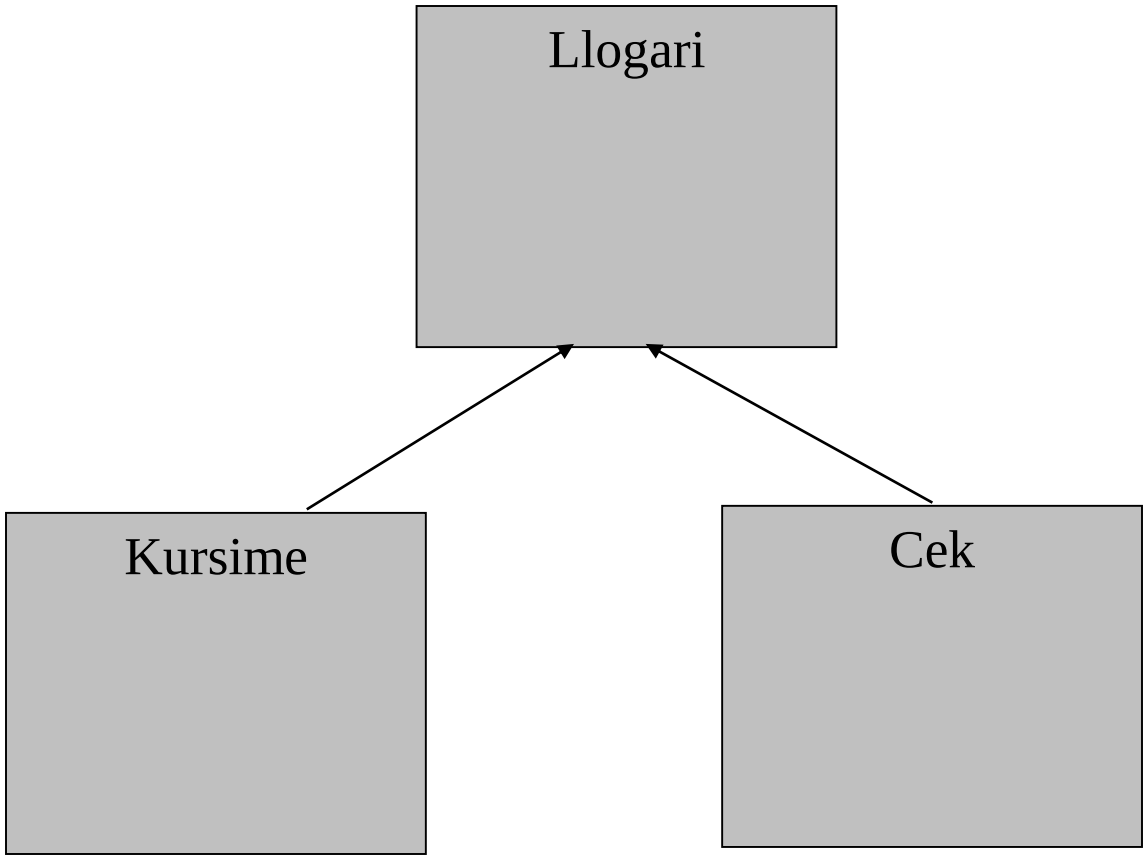
- Më poshtë shihet një klasë B e trashëguar nga klasa A

```
class A { }
class B : A { }
```

Një klasë A dhe nënklasa e saj B

Klasa B trashëgohet nga klasa A

Superklasa Llogari dhe nenklasat Kursime dhe Cek



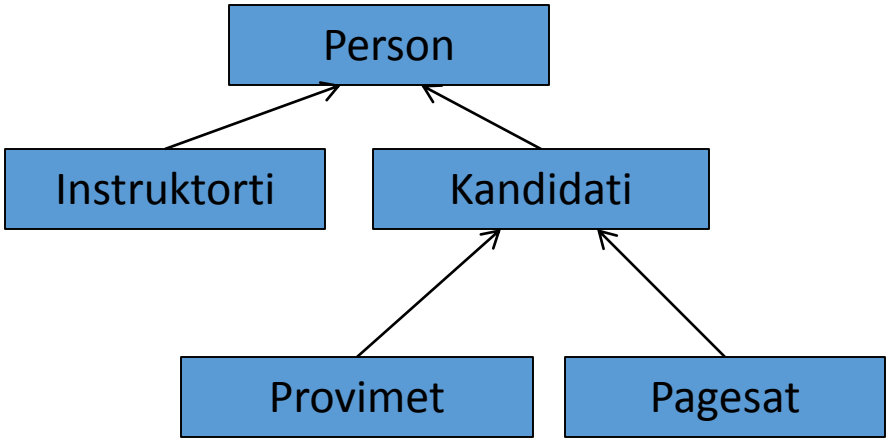
Trashigimia (hiarkia) e klasave

• Një mundësi e tillë e krijimit të klasave të reja, si prej klasave bazë ashtu edhe prej klasave të nxjerra.

•Përmes mekanizmit të trashëgimisë, paraqet një *organizim hierarkik të klasave,* gjë që programimin e orientuar në objekte e bën edhe më të fuqishëm.

• Organizimi i tillë hierarkik, p.sh., mund të duket si ai që është dhënë në figurën e më poshtme, ku prej klasës bazë **Person** janë nxjerrë klasat **Instruktori** dhe **Kandidati.**

• Pastaj, prej klasës **Kandidati** janë nxjerrë klasat **Provimet** dhe **Pagesat.**



Dukja e një organizimi hierarkik të klasave

Specializimim i Klasave

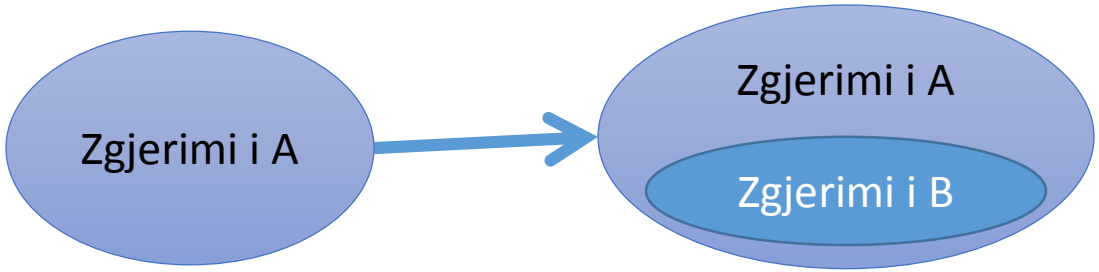
Klasat konsiderohen *tipe*, specializimet si *nën-tipe.*

Specializimi ndihmon në definimin e klasave të reja nga klasat ekzistuëse në bazë të shëndoshë konceptuale.

- Nëse klasa B është specializim i klasës A, atëherë:
  - Instancat e B janë nënbashkësi të instancave të A
  - Operacionet dhe variablat në klasën A janë prezente edhe në B
  - Disa operacione në A mund të ridefinohen në B

Zgjerimi i specializimit të klasave

Zgjerimi (extension) i klasës është nënbashkësi e *zgjerimit* të klasës së përgjithësuar A

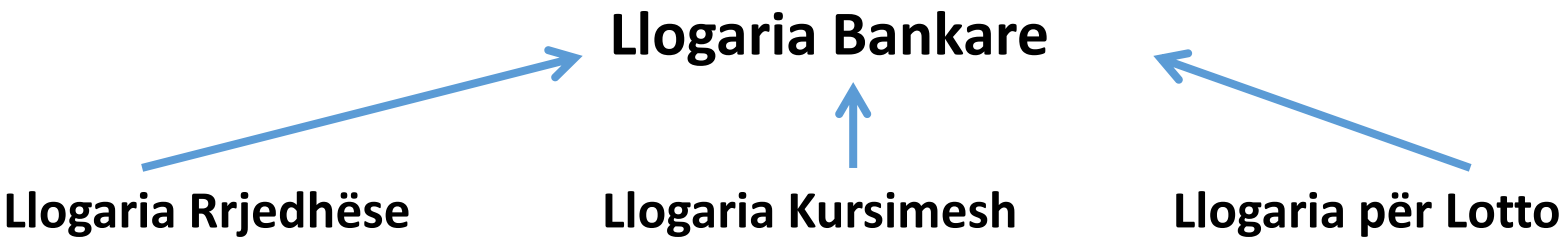


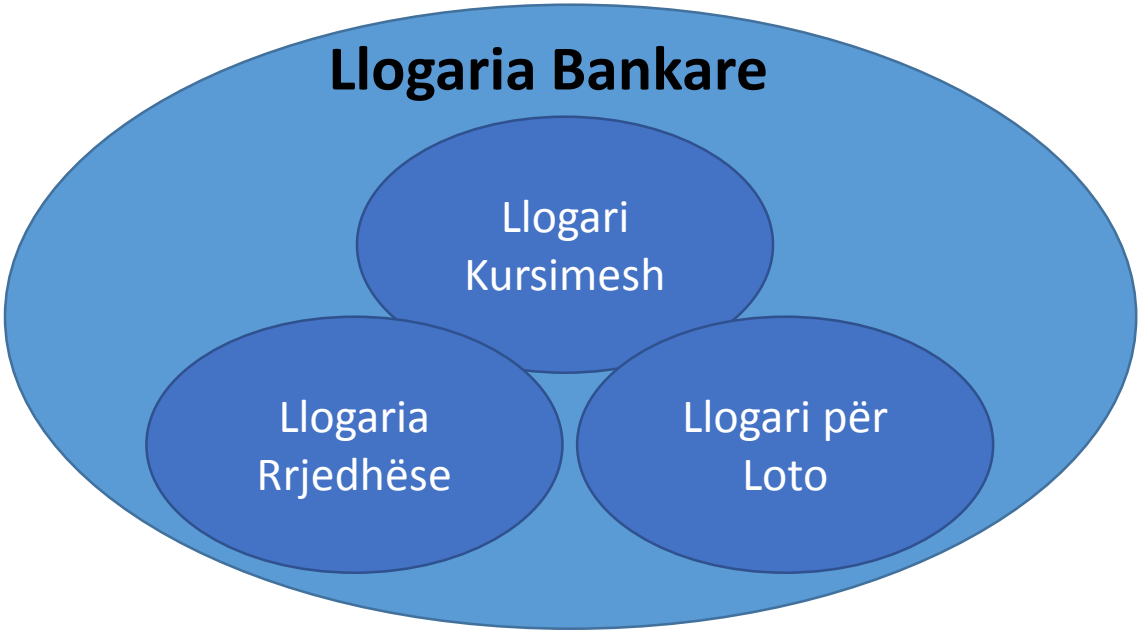
- Relacioni ***is-a (është)***
  - Objekti B është (is-an) Objekt A
  - Ka lidhje në mes të klasës A dhe B

Relacioni **is-a *(është)*** formon një kontrast të relacionit **has-a (ka)**

is-a e karakterizon *specializimin*
has-a e karakterizon *agregimin*

Shembull: Llogaria Bankare





Llogaria Bankare në C#

Llogaria Rrjedhëse **është** (is-a) LlogariBankare, Llogaria e Kursimeve **është** LlogariBankare, Llogaria për Loto **është** LlogariBankare

- Klasa bazë LlogariaBankare
- Klasa LlogariRrjedhese
- Klasa LlogariKursimesh

Specializimi i objekteve të klasave

- Objektet e klasave të specializuara
  - Përfshijnë kushte më të fuqishme (kondita) se objektet e klasave të gjeneralizuara
    - U binden invariancave më të fuqishme të klasave
  - Kanë operacione më të thjeshta dhe më të sakta se objektet e klasave të gjeneralizuara

Principi i Zëvendësimit

Kur specializimi përdoret në formë të pastër aplikohet *principi i zëvendësimit*

- Nëse B është nënklasë e A, është e mundshme që të zëvendësojmë instancën e dhënë B me atë të A pa ndonjë efekt të dukshëm

- Shembull konkret:
  - Është e mundur të ndryshojmë LlogariaBankare me LlogariRrjedhëse
  - Në përgjithësi, nuk është e mundshme të ndryshojmë LlogariaRrjedhëse me LlogariaBankare

Zgjerimi i Klasave

Klasat mund të mendohen si tipe dhe module.
Zgjerimi i klasave është një mekanizëm i *transportim të programit* dhe *rishfrytëzim i programit*

- Nëse B është *zgjerim* i klasës A, atëherë
  - B mund të shtoj variabla dhe operacione të reja në A
  - Operacionet dhe variablat në A janë gjithashtu prezente në B
  - Objektet e B nuk janë domosdoshmërisht të lidhura konceptualisht me Objektet e A

Një shembull i një zgjerimi të thjeshtë

Zgjerojmë klasën Pika2D në klasën Pika3D

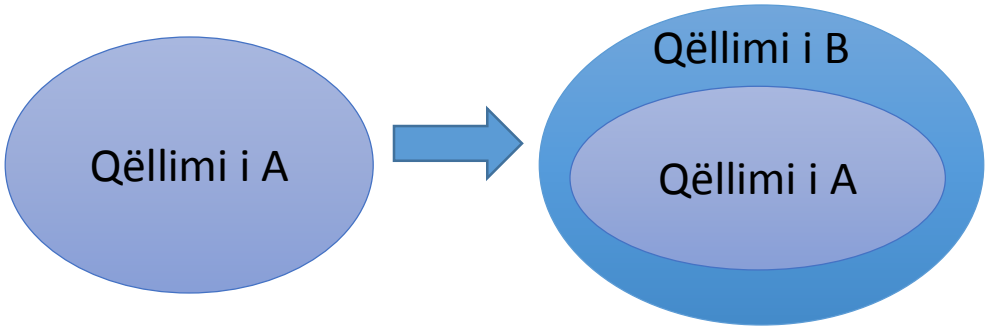
- Klasa Pika2D
- Klasa Pika3D e cila e zgjeron klasën Pika2D
- Klienti i klasave Pika3D dhe Pika2D

- Disa vështrime:
  - Pika 3D nuk është Pikë 2D
  - Kështu, *Pika3D* nuk është specializim i klasës *Pika2D*
  - Bashkësia e objektëve pikë 2D është disjunkte nga bashkësia e pikave 3D

Klasa Pika2D ishte pikë e përshtatshme e klasës Pika3D
Ne i kemi ripërdorur disa të dhëna dhe operacione nga klasa Pika2D në klasën Pika3D

Qëllimi i zgjerimit të klasave

Qëllimi i zgjerimit të klasës B është superbashkësi e qëllimit të klasës origjinale A

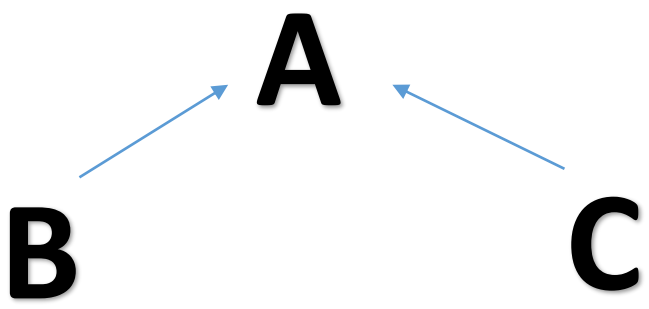


Në përgjithësi, nuk është e mundur të karakterizojmë *zgjerimin* e B në lidhje me *zgjerimin* A

Shpesh, *zgjerimi* i A nuk preket me zgjerimin e B

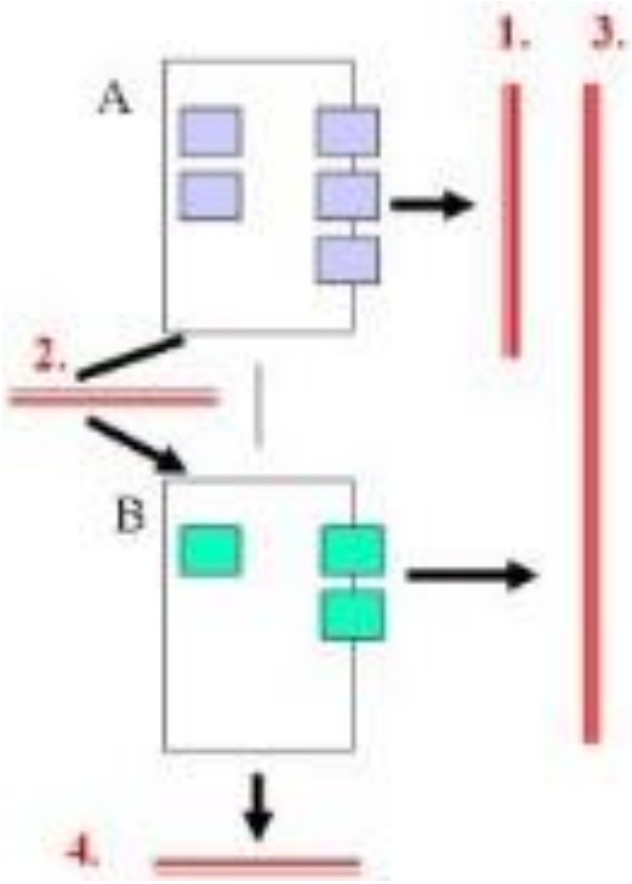
Trashëgimia në përgjithësi

Trashëgimia është një mekanizëm në gjuhët programuëse të orientuara në objekte e cila e lehtëson *specializimin e klasave* dhe *zgjerimin e klasave*



- **Trashëgimia e B dhe C nga A:**
  - Organizon klasët në hierarki
  - Ofron shkallë të specializimit dhe/ose të zgjerimit të A
  - Gjatë kohës së zhvillimit, të dhënat dhe operacionet në A mund të ripërdoren në B dhe C *pa kopjuar* dhe pa ndonjë dyfishim në programin burimor
  - Gjatë kohës së ekzekutimit, instancat e klasës B dhe C janë objekte të komplete, pa pjesët e A

Ndërfaqet tek klientët dhe nënklasat



Ndërfaqet në mes të A , B, klasave klientë të tyre, dhe nënklasave

1. Klient ndërfaqia e A
2. Ndërfaqia nënklasë në mes A dhe nënklasës së saj B
3. Ndërfaqia nënklasë në mes B dhe nënklasës potenciale B

Dukshmëria dhe Trashëgimia

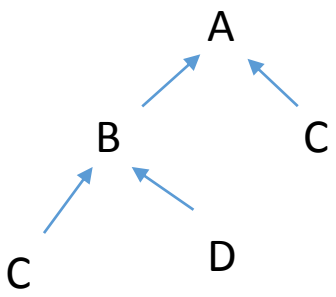
Në shumicën prej gjuhëve programuëse të orientuara në objekte është e mundshme të kontrollohet dukshmëria e të dhënave dhe operacioneve në lidhje me nënklasat

- Veçoritë e dukshmërisë:
  - **Private**
    - Dukshmëria e limituar tek vet klasa
    - Instancat e klasës së dhënë mund të shohin të dhënat private të njëra tjetrës dhe operacionet
  - **Protected**
    - Dukshmëria është e limituar tek vet klasa dhe te nënklasat
  - **Public**
    - Nuk ka kufizime në dukshmëri

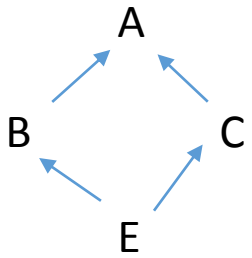
Hierarkia e klasave dhe Trashëgimia

Relacionet e trashëgimisë në mes të bashkësive të klasave definojnë strukturën grafike të klasave.

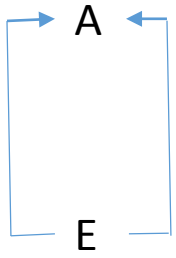
Ky graf strukturorë, është në përgjithësi, aciklik.



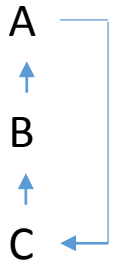
Gjithmonë në rregull



Trashëgimia e shumëfishtë. Në rregull në disa gjuhë



Trashëgimia e përsëritur. Në rregull në disa gjuhë



Gjithmonë ilegale!

Trashëgimia e shumëfishtë **dhe** ***trashëgimia e përsëritur*** **lejohen në disa gjuhë programuëse të orientuara në objekte**

Trashëgimia e shumëfishtë

Pse duhet të kujdesemi për trashëgiminë e shumëfishtë?

- Specializimi i dy e më shumë klasave
  - ***Shembull***:

    Trekëndëshi i djathtë barabrinjësh **është** trekëndësh barabrinjësh dhe **është** trekëndësh i drejtë
  - ***Shembull****:*

    Mund të ketë një llogari bankare e cila **është** llogari e kursimeve dhe **është** llogari rrjedhëse

  - Zgjerimi i dy e më shumë klasave
    - "Transport i programit" nga shumë super klasa

Trashëgimia e klasa në C#

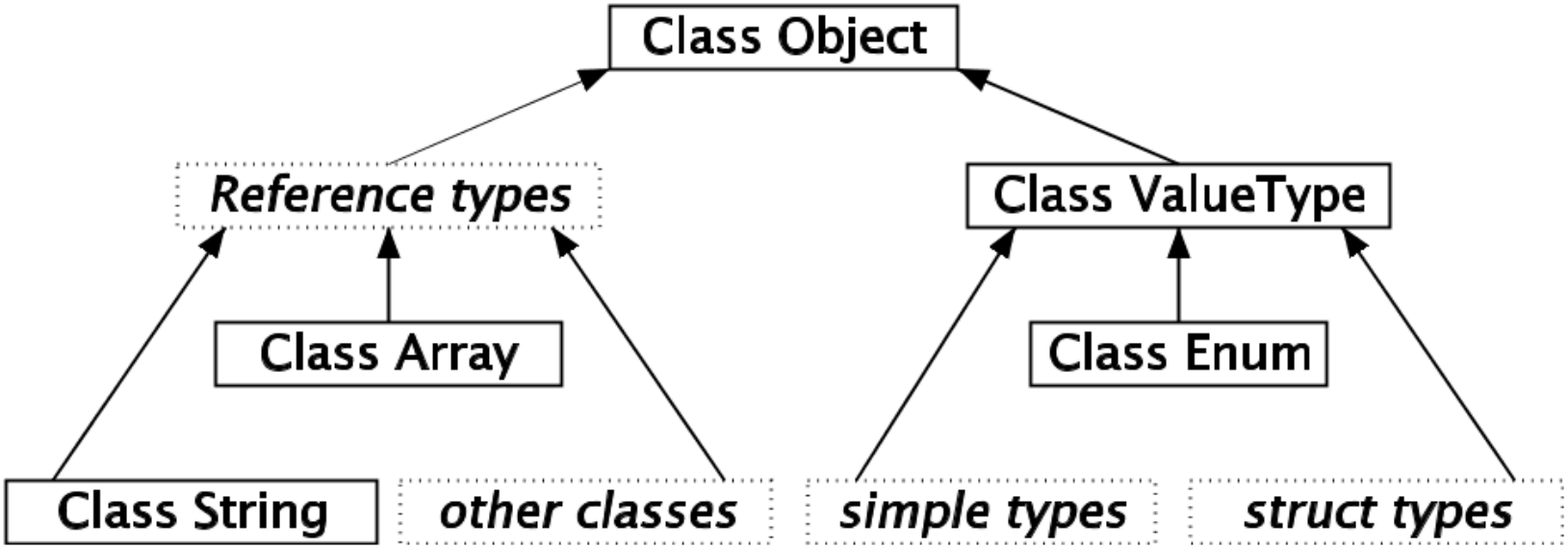
class-modifier **class** *emri-klasës*: *emri-i-super-klasës*{ *deklarimet* }

```
class A {}
class B: A {}
```

B thuhet të jetë *nënklasë* e A, dhe A është *superklasë* e B. A gjithashtu thirret edhe si *klasa bazë* e B.

Rrënja e majes së hierarkisë

*Rrënja e majës së hierarkisë është klasa e quajtur **Object***
Metodat në klasën **Object** ndahen në mes të tipeve referencore dhe atyre me vlerë



Metodat në klasën object në C#

*Rrënja e majës së hierarkisë është klasa e quajtur **Object***
Metodat në klasën **Object** ndahen në mes të tipeve referencore dhe atyre me vlerë

- Metodat publike në klasën object
  - Equals:
    - Obj1.Equals(Obj2) - metodë e instancës
    - Object.Equals(obj1, obj2) - Metodë statike
    - Object.ReferenceEquals(obj1, obj2) – Metodë statike
  - Obj.GetHashCode()
  - Obj.GetType()
  - Obj.ToString()
  - Protected Metodat në klasën Object
  - Obj.Finalize()
  - Obj.MemberwiseClone()

Trashëgimia dhe Konstruktorët

Konstruktorët nuk trashëgohen

- Secila klasë në hierarkinë e klasave duhet të ketë konstruktor(ët) e saj
- Konstruktori i klasës C kooperon me konstruktorin e supërklasës C për të inicuar instancën e re të C
- Konstruktori në nënklasë gjithmonë, në mënyrë implicite apo eksplicite, do t'i referohet konstruktorit të supërklasës së tij
- Konstruktorët në klasën LlogariaBankare
- Konstruktorët në klasën LlogariaKursimeve

Konstruktorët dhe rënditja e inicimit

Variablat e instancës së objektit të ri të alokuar inicohen në mënyrë (rend) të caktuar

- Inicializimi i objektit C: **new C(...)**
  - Variablat e instancës së C janë inicuar (inicimi i fushave)
  - Variablat e instancës në superklasën janë inicuar - më të specializuarat së pari
  - Konstruktorët e supërklasës ekzekutohen – më të përgjithsuarit së pari
  - Trupi i konstruktorit të C , është ekzekutuar

Konstruktorët dhe rënditja e inicimit: Shembull

Instancimi i klasës C që trashëgon klasën B e cila trashëgon klasën A

```
    public class A
    {
        private int varA1 = Init.InitMe(1, "varA1,
inicializues në klasën A"),
       varA2;

        public A()
        {
            varA2 = Init.InitMe(4, "VarA2,trupi i
 konstruktorit A");
        }
    }
    public class B : A
    {
        private int varB1 = Init.InitMe(1, "varB1,
inicializues në klasën B"),
                    varB2;
        public B()
        {
            varB2 = Init.InitMe(4, "VarB2, trupi i
```

```
 public class C : B
    {
        private int varC1 =
Init.InitMe(1, "varC1, inicializues
në klasën C"),
                    varC2;

        public C()
        {
            varC2 = Init.InitMe(4,
"VarC2, trupi i konstruktorit  C");
        }
    }

    public class Init
    {

        public static int InitMe(int
val, string kush)
        {
            Console.WriteLine(kush);
            return val;
        }

    }
```

Modifikatorët e dukshmërisë në C#

Dukshmëria e tipeve dhe në asambler dhe anëtarëve në klasa kontrollohen nga modifikatorët e dukshmërisë

- Dukshmëria e tipeve
  - **Internal**: tipi nuk është i dukshëm jashtë asamblerit
  - **public:** tipi është i dukshëm jashtë asamblerit
- Dukshmëria e tipeve anëtare (p.sh., metodat në klasë)
  - **`private:`** qasja vetëm nga brenda klasës
  - **`protected:`** qasje në tipin përmbajtës dhe nëntipin
  - **`internal:`** qasje në asambler
  - **`protected internal:`** qasje në asambler dhe tipin përmbajtësor dhe nëntipet e tij
  - **`public:`** i qasshëm kurdo që tipi është i qasshëm

Trashëgimia e metodave, vetive, dhe indeksave

Metodat, vetitë dhe indeksat trashëgohen

- Metodat, vetitë dhe indeksat mund të ridefinohen në dy mënyra:
  - Emra të njejtë dhe nënshkrim në super dhe nënklasë, me kuptime të ngjashme (virtual , override)
  - Emra të njëjtë dhe nënshkrime në super dhe nënklasë, dy kuptime komplet të ndryshme (new)
- Metoda M në klasën B mund t'i referohet metodës M në klasën A
  - base.M(...)
  - Kooperimi, i njohur edhe si kombinim i metodave

Operatorët trashëgohen.
Operatori i ridefinuar në supërklasë do të jetë operator i tërë i ri .

Mbingarkimi dhe Fshehja në C#

Çka nëse metoda M në klasën A paraqitet edhe në klasën B ?

```
class A
{
    public void M() { }
}
class B : A
{
    public void M() { }
}
```

- Ridefinimi i kërkuar:
- B.M kërkohet të ridefinoj A.M ashtu që B.M të përdoret në instancat e B
  - A.M duhet të deklarohet **virtual**
  - B.M duhet të deklarohet me **override** A.M
- Ridefinimi Aksidental

  Programeri i klasës B nuk vëren A.M
  - B.M duhet deklaruar që nuk është në lidhje me A.M duke përdorur modifikatorin **new**

Polimorfizmi. Tipet Statike dhe Dinamike

Polimorfizmi dhe shfrytëzimi i përshtatshëm i lidhjes dinamike është *kupa e artë të OOP* në lidhje me trashëgiminë

- Polimorfizmi qëndron prapa idesë që një variabël mund t'u referohet disa tipeve të ndryshme

- Tipi static i variablës është tip i variablës siç është deklaruar

- Tipi dynamic i varablës është tip i objektit të cilit i referohet variabla

Lidhja Dinamike është në efekt nëse tipi dinamik i variablës **v** përcakton operacionin e aktivizuar nga **v.op(...)**

Lidhja Dinamike arrihet përmes metodave virtuale

Tipet dinamike dhe statike në C#

Njohuria mbi tipet statike të variablave dhe shprehjeve shfrytëzohet për *kontrollimin në kohën e kompajllimit të tipeve*

```
class A { }
    class B : A { }

    class Klienti
    {
        public static void Main()
        {
            //tipi Static   tipi Dynamic
            A x;          //    A             -
            B y;          //    B             -

            x = new A(); //    A             A   TRIVIAL
            y = new B(); //    B             B   TRIVIAL

            x = y;       //    A             B   OK- Tipik

            y = new A(); //    B             A   GABIM gjatë kohës së kompajlimit
            //                                 Cannot implicitly convert type 'A' to 'B'.
            y = x;       //    B             B   GABIM gjatë kohës së kompajlimit
            //                                  Cannot implicitly convert type 'A' to 'B'.
```

Testimi i tipeve dhe konvertimi në C#

Është e mundshme të testohet nëse tipi dinamik i variablës v është klasë e tipit C
Janë dy mënyra të konvertimit (cast) një tip të një klase në një tjetër

v është c

- E vërtetë nëse variabla v është e *tipit dinamic* c
- Gjithashtu e vërtetë nëse varibla v është e tipit dinamic D, ku D është nëntip i c
- (C)v
  - Konverto tipin *static* të v në c me shprehjen e dhënë
  - Është e mundur vetëm nëse tipi dinamik v është c, ose nëntip i c
  - Nëse jo, gabimi ***InvalidCastException*** do të hidhet
- v sikurse c
  - Variant jo-fatal i (C)v
  - Kështu, konverton tipin static të v në c në shprehjen e dhënë
  - Kthen null nëse tipi dinamic i v nuk është c, ose nëntip i c

LlogariaBankare.cs

```
using System;

public class LlogariaBankare
{

    protected double interesi;
    protected string pronari;
    protected decimal bilanci;

    public LlogariaBankare(string o, decimal b, double
ir)
    {
        this.interesi = ir;
        this.pronari = o;
        this.bilanci = b;
    }

    public LlogariaBankare(string o, double ir) :
        this(o, 0.0M, ir)
    {
    }
```

```
    get { return bilanci; }
}

public virtual void Terheqje(decimal shuma)
{
    bilanci -= shuma;
}

public virtual void Depozito(decimal shuma)
{
    bilanci += shuma;
}

public virtual void ShtoInteres()
{
    bilanci += bilanci * (Decimal)interesi;
}

public override string ToString()
{
    return pronari + " ka  " +
```

LlogariaRrjedhese.cs

```
using System;

public class LlogariaRrjedhese: LlogariaBankare {

   public LlogariaRrjedhese(string o, double ir):
     base(o, 0.0M, ir) {
   }

   public LlogariaRrjedhese(string o, decimal b, double ir):
     base(o, b, ir) {
   }

   public override void Terheqje(decimal shuma)
   {
      bilanci -= shuma;
      if (shuma < bilanci)
         interesi = -0.10;
   }

   public override string ToString() {
      return pronari + " i llogarise rrjedhese ka " +
            + bilanci + " euro";
   }
}
```

LlogariKursimesh.cs

```
public class LlogariKursimesh : LlogariaBankare
    {
        public LlogariKursimesh(string o, double ir) :
            base(o, 0.0M, ir)
        {
        }

 public LlogariKursimesh(string o, decimal b, double ir) :
            base(o, b, ir)
        {
        }

        public override void Terheqje(decimal shuma)
        {
            if (shuma < bilanci)
                bilanci -= shuma;
            else
         throw new Exception("Nuk mund te terhiqni!");
        }

        public override void ShtoInteres()
        {
            bilanci = bilanci + bilanci *
(decimal)interesi - 100.0M;
        }

        public override string ToString()
        {
            return pronari +
                " i llogarise se kursimeve ka " +
                  +bilanci + " euro";
        }
    }
```

Pika2D.cs

```
public class Pika2D
    {
        private double x, y;

        public Pika2D(double x, double y)
        {
            this.x = x; this.y = y;
        }

        public double X
        {
            get { return x; }
        }

        public double Y
        {
            get { return y; }
        }

        public void Levize(double dx, double dy)
        {
            x += dx; y += dy;
        }

        public override string ToString()
        {
            return "Pika2D: " + "(" + x + ", " + y + ")" +
                                  ".";
        }
    }
```

Pika3D.cs

```
public class Pika3D : Pika2D
    {

        private double z;

        public Pika3D(double x, double y, double z) :
            base(x, y)
        {
            this.z = z;
        }

        public double Z
        {
            get { return z; }
        }

        public void Levize(double dx, double dy, double
dz)
        {
            base.Levize(dx, dy);
            z += dz;
        }

        public override string ToString()
        {
            return "Pika3D: " + "(" + X + ", " + Y + ", "
+ Z + ")" + ".";
        }
    }
```

Programi me klasat Pika2D dhe Pika3D

```
class Program
   {
       static void Main(string[] args)
       {

           Pika2D p1 = new Pika2D(1.1, 2.2),
           p2 = new Pika2D(3.3, 4.4);

           Pika3D q1 = new Pika3D(1.1, 2.2, 3.3),
           q2 = new Pika3D(4.4, 5.5, 6.6);

           p2.Levize(1.0, 2.0);
           q2.Levize(1.0, 2.0, 3.0);
           Console.WriteLine("{0} {1}", p1, p2);
           Console.WriteLine("{0} {1}", q1, q2);
       }
   }
```

Konstruktori në klasën Llogaria Bankare

```
public class LlogariaBankare
    {

        protected double interesi;
        protected string pronari;
        protected decimal bilanci;

        public LlogariaBankare(string o,
decimal b, double ir)
        {
            this.interesi = ir;
            this.pronari = o;
            this.bilanci = b;
        }

        public LlogariaBankare(string o,
double ir) :
            this(o, 0.0M, ir)
        {
        }
    }
```

Konstruktori në klasën LlogariaKursimeve

```
 public class LlogariaKursimeve :
LlogariaBankare
    {


        public LlogariaKursimeve(string
o, double ir) :
            base(o, 0.0M, ir)
        {
        }


        public LlogariaKursimeve(string
o, decimal b, double ir) :
            base(o, b, ir)
        {
        }
    }
```

Përmirësimi i gabimeve

```
class A { }
   class B : A { }
   class Client
   {
       public static void Main()
       {
           // tipi Static  tipi Dynamic
           A x;            //    A              -
           B y;            //    B              -

           x = new A();    //    A       A TRIVIAL
           y = new B();    //    B       B TRIVIAL

           x = y;          //    A    B   OK - Tipik

           y = (B)new A(); //  B   A   RUNTIME ERROR
           y = (B)x;       //  B   B   Tani OK
       }
   }
```